Coping with NP-Hardness

- Strategies to deal with hard problems
- Algorithms for Special Cases
- Fixed Parameter Tractability
- Intelligent Exhaustive Search
  - Backtracking
  - Branch and Bound
- Dynamic Programming based pseudo polynomial algorithm TSP

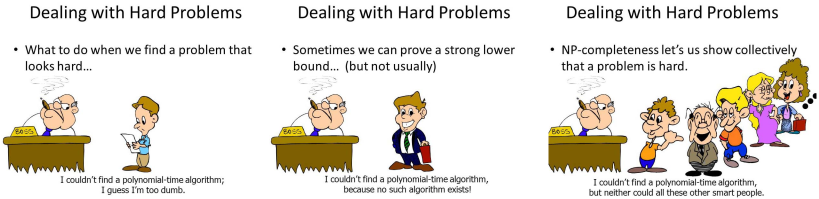
Imdad ullah Khan

Intractable Problems in Practice

Try to solve a problem through some design paradigm

If fruitless, try to prove that your problem is NP-Hard

Good theoretical exercise, but the problem doesn't go away



source: slideplayer.com via Google images

In this lecture we briefly explore what to do in this case

NP-Completeness is not a death certificate, it is the beginning of a fascinating adventure

NP-Hardness

When you prove a problem X to be NP-Hard, then as per the almost consensus opinion of $\mathrm{P} \neq \mathrm{NP}$, it essentially means

1. There is no polynomial time
2. deterministic algorithm
3. to exactly/optimally solve the problem X
4. for all possible input instances

What are the option? Things to consider when your problem is NP-Hard

Coping with NP-HARDNESS

- Do I need to solve the problem for all valid input instances?
  - Sometimes just need to solve a restricted version of the problem -
    ▷ (special cases) that include realistic instances
- Is exponential-time OK for my instances?
  - Exponential-time algorithms are "not slow" ▷ they don't scale well
  - If relevant instances are small, then they may be acceptable
  - Can bring exponent/base of runtime down ▷ $2^n \rightarrow 2^{\sqrt{n}}$ or $2^n \rightarrow 1.5^n$
- Is non-optimality OK?
  - What if our algorithm is better than others ▷ faster than bruteforce

Coping with NP-Hardness

To cope with NP-Hardness, sacrifice one of these features

Poly-time	Deterministic	Exact/Opt Solution	All cases/ Parameters	Algorithmic Paradigm
✓	✓	✓	✗	Special Cases Algorithms Fixed Parameter Tractability
✓	✓	✗	✓	Approximation Algorithms Heuristic Algorithms
✗	✓	✓	✓	Intelligent Exhaustive Search
✓	✗	$\mathbb{E}$(✓)	✓	Mote Carlo Randomized Algorithm
$\mathbb{E}$(✓)	✗	✓	✓	Las Vegas Randomized Algorithm

- Special cases of input instances (based on structure of a range of parameter(s))
- Approximation algorithms guarantee a bound on suboptimality
- Heuristics algorithms do not have any guarantee
- Randomized algorithms are generally used for problems in class P

Approaches to tackle hard problems

1. Special Cases: Relevant structure on which the problem is easy
   - Exact results in poly-time only for special cases or a range of parameters
2. Intelligent Exhaustive Search: Exponential time in worst case
   - The base and/or exponent are usually smaller
   - Could be efficient on typical more realistic instances
   - Backtracking, Brand-and-Bound
3. Nearly exact solutions: Output is *'close'* to exact (optimal) solution
   - Approximation Algorithms: Solutions of guaranteed quality in poly-time
   - Heuristic: Solutions hopefully good in poly-time
4. Randomized Algorithms: Use coin flips for making decisions
   - Typically used for approximation, also used for problems in P